

DEVELOPING INTERACTIVE PARALLEL VISUALIZATION FOR COMMODITY-BASED CLUSTERS

S.TOMOV , R.BENNETT , M.MCGUIGAN , A.PESKIN , G.SMITH , AND J.SPILETIC*

November 8, 2002

Abstract. We present an efficient and inexpensive solution for developing interactive high-performance parallel visualization. The proposed solution is used to extend popular APIs such as Open Inventor [16, 17] and VTK [13, 14] to support commodity-based cluster visualization. Our implementations have several variations, but in general the idea is to have a “Master” node, which will intercept a sequential graphical user interface (GUI) and broadcast it to the “Slave” nodes. The interactions between the nodes are implemented using MPI [5]. The parallel remote rendering uses Chromium [7]. We present in detail the proposed model and key aspects of its implementation. Finally, we present performance measurements that show that the suggested idea is feasible and yields good results in the development of interactive parallel visualization.

Key words. Interactive visualization, visualization APIs, commodity-based clusters visualization, parallel visualization, remote rendering, Chromium, Open Inventor, VTK, MPI.

1. Introduction. We are interested in large scale high-performance scientific parallel visualization with real-time interaction. More precisely, we want to support the interactive extraction of insight from extremely large sets of data. Also, in order not to miss important details in the data, we would like to develop high resolution visualization. Our interest is motivated by today’s technological advances, which stimulate and facilitate the development of increasingly complicated mathematical models. Visualization with real-time interaction is essential for better analysis and understanding of the masses of data produced by such models, or by devices such as Magnetic Resonance Imaging (MRI) and laser-microscope scanners. Some of the applications that we are interested in and work on are given on Figure 1.1. Others come from high-energy physics, climate modeling, etc. Currently the size of the data sets that we are dealing with is around 1 GB.

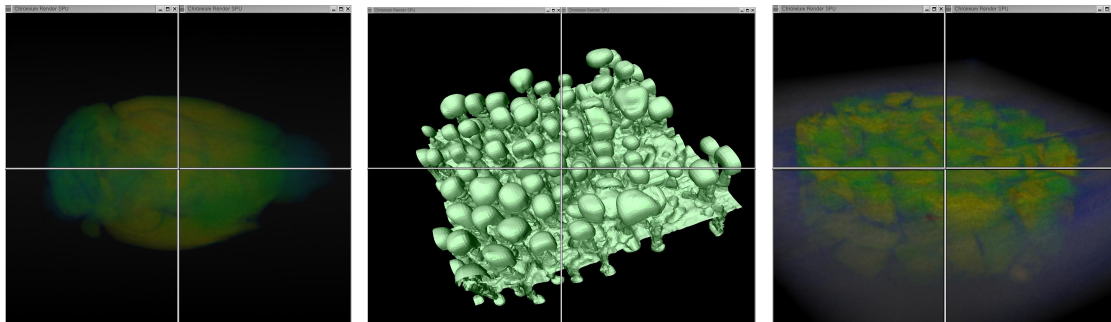


FIG. 1.1. *Left: MRI mouse brain volume visualization on a 512 X 256 X 256 grid; Middle: Fluid flow isosurface visualization; Right: Material micro-geometry studies: crushed rock volume visualization on a 1007 X 1007 X 256 grid. The three examples are rendered in parallel on a display composed of 4 tiles.*

We propose an inexpensive and, as the results show, efficient solution to the visualization problem outlined above. The approach is to

- do remote parallel rendering to a large tiled display (see for example Figure 1.1 where we show displays composed of 4 tiles);

*Information Technology Division, Brookhaven National Laboratory, Bldg. 515, Upton, NY 11973
tomov@bnl.gov, robertb@bnl.gov, mcguigan@bnl.gov, peskin@bnl.gov, smith3@bnl.gov, spiletic@bnl.gov

- use commodity-based clusters connected with high speed network;
- extend and combine already existing APIs such as Chromium, Open Inventor, VTK, etc.

The APIs considered are open source. Open Inventor (see [16, 17]) is a library of C++ objects and methods for building interactive 3D graphics applications, VTK (see [13, 14]) is another widely used API for visualization and image processing, and Chromium (see [7, 2, 3, 6]) is an OpenGL [12] interface for cluster visualization to a tiled display. Chromium is based on WireGL [8] and provides a scalable display technology.

There are feasible alternatives to our approach. For example, the tiled display can be replaced with a high resolution flat panel driven by a single workstation, the visualization clusters can be replaced with fast sequential machines, and the process of extending and combining already existing APIs can be replaced with building the entire visualization system from scratch. Extending and combining visualization APIs is a popular and in many cases preferred development approach. It is appealing because one can easily leverage already existing and powerful APIs. Examples are the extension of VTK to ParaView (see [10]) and parallel VTK (see [1]), **VisIt** (see [4]), etc. Building an entire visualization system from scratch may be efficient for very particular requirements, but in general is expensive and time consuming (see for example NASA’s long term project **ParVox** [11]). Our approach serves well the visualization needs and goals that we currently have, it is inexpensive and easy to develop, and it works, which is shown by extending the Open Inventor and the VTK APIs.

The parallel model that we consider is MIMD (multiple instructions – multiple data). The cluster nodes run OpenGL visualization of separate parts of a global scene. Chromium is used for compositing their output to a large tiled display. MIMD parallel models are common for clusters of workstations and to provide them with user interaction is a task of great interest (see for example [15]). We pursue two approaches. The first one is to declare one of the cluster nodes as Master, which will be used to intercept the sequential user input and broadcast it in a user defined protocol to the other nodes (here called Slave nodes). This yields a system where a Master sends to the Slave nodes instructions of “how and when” to redraw their part of the scene. The other interaction style is to have a separate “parallel interactor” device. This would be a GUI window that will intercept the sequential user input and broadcast it to the cluster nodes through sockets. The details are given in the following sections.

The article is organized as follows. In Section 2 we describe the proposed model and its implementation in extending Open Inventor and VTK. Next (in Section 3) we discuss issues that are important for the development of high performance visualization on clusters of workstations. Also, we provide some of our performance results. The last section (Section 4) summarizes the results of this work.

2. Parallel GUI model and implementation. Parallel GUI for a MIMD parallel visualization model requires the presence of a Master node that will synchronize with the other nodes the redrawing of the consecutive scenes according to a sequential user input. As mentioned in the introduction, we consider two variations of parallel GUI. Both have similar visualization pipelines (see Figure 2.1). The parallel GUI (ParaMouse) gets the sequential user input (through mouse, keyboard, etc.) and broadcasts it to the OpenGL applications through sockets. Every application is visualizing a separate part of a global scene. The applications are bound together by MPI communications. The OpenGL calls that the applications make are intercepted by Chromium and sent through the network to the visualization servers, which composite the input and render it to a tiled display.

Chromium’s parallelization model (denoted in the article by CPM) is represented with the pseudo-code:

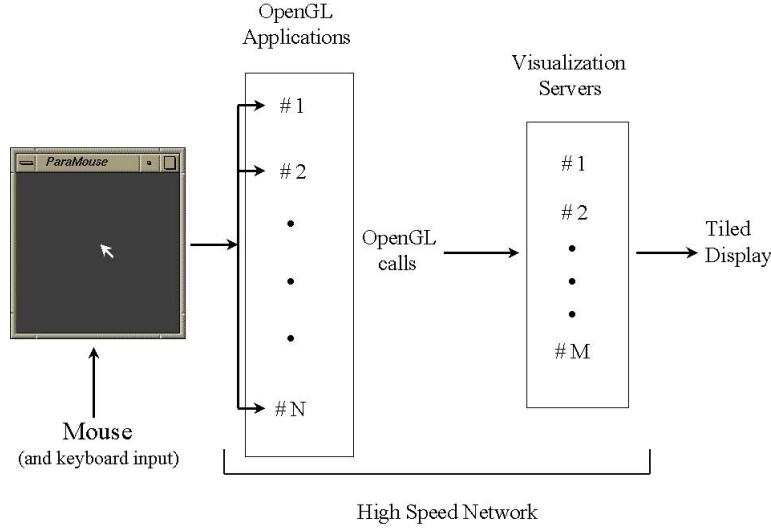


FIG. 2.1. *Visualization pipeline. The parallel GUI (ParaMouse) gets the sequential user input (through mouse, keyboard, etc.) and broadcast it to the OpenGL applications through sockets. The applications use Chromium to render to a large tiled display.*

```

glXMakeCurrent(getDisplay(), getNormalWindow(), getNormalContext());
if (clearFlag)
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glBarrierExecCR(MASTER_BARRIER);

do sequential OpenGL rendering of the local scene

glBarrierExecCR(MASTER_BARRIER);
if (swapFlag)
    glXSwapBuffers(getDisplay(), getNormalWindow());
else
    glXSwapBuffers(getDisplay(), CR_SUPPRESS_SWAP_BIT);
  
```

where the display obtained is **composited** if `clearFlag` is 1 for all processors, `swapFlag` is 1 for processor with rank 0, and 0 for the rest, and **tiled** if `clearFlag` and `swapFlag` are 1 only for processor with rank 0.

To extend Open Inventor and VTK (or any API) to support interactive cluster visualization within the above framework we

- apply the CPM to the APIs rendering method(s);
- implement the ParaMouse parallel GUI or extend the API's GUI using the Master-Slave concept.

To implement the first step for Open Inventor we extended the `SoXtRenderArea::redraw()` method. For VTK this step is implemented by D.Thompson, Sandia National Laboratory. We implemented in Open Inventor the Master-Slave concept by extending the GUI that `ivview` provides.

Function `main` in `ivview` was extended by implementing the pseudo-code:

```
declare processor with rank 0 as Master;
if (Master)
    run GUI as implemented in ivview;
else
    listen for instructions from the Master;
```

We extended the callback functions triggered by the devices that Open Inventor’s GUI handles (mouse, keyboard, etc). The callback functions responding to user input have `MPI_Isend` of the invoked user input in the Master node to the Slave nodes. The Slave nodes are in “listening” mode (`MPI_Recv` or `MPI_Irecv` while animating), which is given with the pseudo-code below, and upon receiving data, representing instructions in an internally defined protocol, they call the action for the invoked input and redraw their part of the global scene (if necessary).

```
// Initialize Inventor
SoDB::init();
SoNodeKit::init();
SoInteraction::init();

// Build the Inventor’s objects and scene graphs
SoSeparator *root = new SoSeparator;
root->ref();
readScene(root, files);

// Create a Slave node ExaminerViewer
SoExaminerViewer *viewer = new SoExaminerViewer(crrank);
viewer->setSceneGraph(root);

// Chromium initialization
crctx = crCreateContextCR(0x0, visual);
crMakeCurrentCR(crwindow, crctx);
glBarrierCreateCR( MASTER_BARRIER, crsize);

glEnable(GL_DEPTH_TEST);
viewer->mainLoop();
```

The `SoExaminerViewer` class is based on the Open Inventor’s `SoXtExaminerViewer` class. The difference is that `SoExaminerViewer` does not have Xt window interface function calls and the rendering is with the CPM.

The user interface is through the `ivview` window (see Figure 2.2, left), which is blank and used only for the user input. The scene is drawn in separate windows/tiles. The example from Figure 2.2 (right) shows a display with 4 tiles. There are 4 applications running, each of which visualize a sphere.

In VTK’s Master-Slave model we extend the `vtkXRenderWindowInteractor` by

- implementing “pure” X Windows GUI (no GL calls);
- adding MPI send/receive calls for mouse and keyboard events.

For the ParaMouse interface we added in VTK a new interactor style,

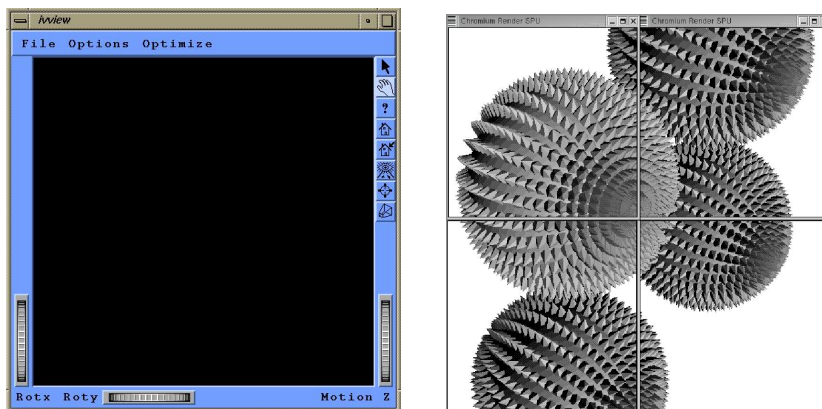


FIG. 2.2. Left: `ivview` window interface; Right: example of a display with 4 tiles. The scene is rendered in parallel on the tiled display. The interactive user interface is through the `ivview` window, with the functionality that `ivview` provides.

called `vtkInteractorStylePMouse`.

3. Observations and performance results. We did our implementation and testing on a Beowulf cluster with 4 nodes, each node with 2 Pentium III processors, running at 1 GHz. Every node has Quadro2 Pro graphics card. The nodes are connected into a local area network with communications running through 100 Mbit/sec fast Ethernet or 1 Gbit/sec fiber optic network. More about the general performance of this machine can be found in [9].

The experience that we had in developing interactive parallel visualization is summarized as follows.

1. For the parallel model considered performance is problem and interaction specific.
For example, depending on the data and the user interaction, the entire global scene may be mapped to a single tile of the display. Also, the Chromium bucketing strategy [2, 3, 6] will not work for scenes composed of non-localized consecutive polygons.
2. The GUI communications time is negligible compared to the visualization time, which leads us to the scalability results reported in [2, 3, 6]. This statement is supported by our next observation.
3. We can send approximately 5,000 “small” (less than 100 bytes) messages/sec (see [9]).
4. Chromium automatically minimizes data flow, except for geometry flow (see below).
5. It is advisable to keep small static scenes in display lists (see below).

The network data flow is usually the bottleneck in the visualization pipeline that we consider. Chromium provides several techniques to minimize it. They are: simplified network protocol, bucketing, and state tracking (see [2, 3, 6]). None of these techniques however are intended for the automatic minimization of the geometry flow, which usually is the most expensive component. For example in animation the same objects are drawn without (or with minimal) change from frame to frame. Nevertheless the scene is transmitted through the network for every frame, creating an enormous bottleneck, as shown in the next example.

The following example illustrates items 4 and 5 from the above observations. We use a small static scene composed of spheres with spikes, as the ones shown on Figure 2.2, right. Each sphere is composed of 11,540 triangles. Sequentially, one sphere is drawn by the Open Inventor at a rate of 195 frames/sec, i.e. 2,251,569 triangles/sec. For 2 spheres the rate is 117 frames/sec or 2,709,130 triangles/sec, etc. The speed of visualizing 2 spheres on 2 processors (2 tiles) is approximately

6 frames/sec. The composited visualization (2 processors, 1 tile) is approximately 3 frames/sec. Runs with various problem sizes and parallel visualization configurations give similar results in favor of the sequential execution.

The enormous difference is due to the fact that in the first case the scene resides in the graphics card memory, while in the second the same scene is transmitted for every frame.

The non-automatic mechanism that Chromium provides for acceleration/minimization of the geometry flow is display lists. They are supported by sending the lists to each rendering server, and thus guaranteeing their presence on the server when they are called. Display lists in Open Inventor are created for the parts of the scene that have as root **SoSeparator** nodes with field **renderCaching** turned **ON** or **AUTO** (for more information see [16], pages 224-227). Using display lists we get a speed of 90 frames/sec for visualizing 2 spheres by 2 processors on composited display, and 80 frames/sec on tiled display. Different parallel and display configurations illustrate the same order of improvement. These results are comparable to the sequential ones. The trade-offs are that the process is not automatic and that the display lists are broadcast to all rendering servers.

The above example was also intended to keep the developers' awareness of when to use and what to expect from interactive parallel visualization systems based on remote rendering. To summarize the results we include the following comparison

- Fast sequential visualization often relies on
 - graphics acceleration hardware;
 - data sampling methods;
 - static scenes;
 - data size within the hardware limitations.
- The proposed visualization system (as a remote visualization)
 - does not require expensive graphics hardware;
 - provides high resolution and interactive speed with sequentially slow methods, such as ray casting (usually there is no need of data sampling);
 - facilitates well the visualization of time-varying scenes;
 - can handle extremely large data sets.

4. Conclusions. We developed interactive parallel visualization for commodity-based clusters. The interaction concept was implemented by extending the interaction functionality of already existing and powerful APIs such as Open Inventor and VTK. We rendered to a large tiled display using Chromium.

We achieved our objectives

- (1) to develop high performance, real-time interaction,
- (2) to be able to handle large sets of data,
- (3) to produce high resolution visualization, and finally
- (4) to be inexpensive and easy to develop.

REFERENCES

- [1]J.Ahrens, C.Law, K.Martin, M.Papka, W.Schroeder, *A Parallel Approach for Efficiently Visualizing Extremely Large, Time-Varying Data Sets*, Los Alamos National Laboratory, Technical Report #LAUR-00-1620.
- [2]I.Buck, M.Eldridge, P.Hanharah, G.Humphreys, *Distributed Rendering for Scalable Displays*, Proceedings of Supercomputing, 2000.
- [3]I.Buck, P.Hanharah, and G.Humphreys, *Tracking Graphics State for Networked Rendering*, Proceedings of SIGGRAPH/Eurographics Workshop on Graphics Hardware, August 2000.
- [4]Eric S. Brugger, *Visit: A Component-Based, Parallel Visualization Package*, Eric S. Brugger - Lawrence Livermore National Laboratory, DOE, Computer Graphics Forum 2001, Internet address (accessed on

- 08/2002):
<http://www.emsl.pnl.gov:2080/docs/doecgf2001/abstractlist.html>
- [5] W. Gropp, *Tutorial on MPI: The Message-Passing Interface*, Mathematics and Computer Sci. Division, Argonne National Laboratory, Internet address (accessed on 06/2002):
<http://www-unix.mcs.anl.gov/mpi/tutorial/index.html>.
 - [6] P. Hanrahan, H. Igely, G. Stroll, *The Design of a Parallel Graphics Interface*, Proceedings of SIGGRAPH, 1998.
 - [7] G. Humphreys, *Chromium Documentation, Version BETA*, Internet address (accessed on 06/2002):
http://graphics.stanford.edu/~humper/chromium_documentation/
 - [8] G. Humphreys, M. Eldridge, I. Buck, G. Stroll, M. Everett, P. Hanrahan, *WireGL: A Scalable Graphics System for Clusters*, Proceedings of SIGGRAPH, 2001.
 - [9] V. Mirinavicius, *Investigation of MPI performance in Parallel Linear Algebra Software on Linux Beowulf Supercomputers*, Brookhaven National Laboratory, Technical Report (to appear).
 - [10] Kitware Inc., *ParaView*, Internet address (accessed on 08/2002):
<http://public.kitware.com/>
 - [11] NASA, Jet Propulsion Laboratory, *ParVox, A Parallel Splatting Volume Rendering System*, Internet address (accessed on 08/2002):
<http://alphabits.jpl.nasa.gov/ParVox/>
 - [12] J. Neider, T. Davis, and M. Woo (OpenGL Architecture Review Board), *OpenGL Programming Guide : The Official Guide to Learning OpenGL, Release 1*, Addison-Wesley Publishing Company, Boston, May 1996.
 - [13] W. Schroeder, K. Martin, L. Avila, C. Law *The Visualization Toolkit User's Guide*, Kitware, Inc.
 - [14] W. Schroeder, K. Martin, B. Lorensen, *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, Prentice-Hall PTR.
 - [15] IEEE Visualization 2002, workshop on: *Commodity-Based Visualization Clusters*, Proceedings of IEEE Visualization, 2002.
 - [16] J. Wernecke, *The Inventor Mentor : Programming Object-Oriented 3D Graphics with Open Inventor*, Addison-Wesley Publishing Company, NY, November 1995.
 - [17] *Open Inventor C++ Reference Manual: The Official Reference Document for Open Inventor, Release 2*, Addison-Wesley Publishing Company, January 1994.